Microsoft Azure

MLOps with Azure Machine Learning

Accelerating the process of building, training, and deploying models at scale

Contents

Introduction 3

Understanding the typical ML lifecycle 4

Accelerating your end-to-end ML lifecycle with MLOps 6

Model reproducibility 6

Model validation 10

Model deployment 13

Model retraining 16

Automating MLOps processes 19

Establishing MLOps at organizational scale 20

Conclusion 20

Additional Resources 21

Enterprise organizations in nearly every industry are increasingly making machine learning (ML) initiatives a priority. Machine learning models are at the core of many AI use cases from predictive maintenance and recommendations to anomaly detection and dynamic pricing.

Modern machine learning algorithms and frameworks make it increasingly easy to develop models that can make accurate predictions. But even when you build a model that exceeds expectations, putting it into production has its own share of challenges. Over time, even a once successful model might deteriorate, or perhaps you think you can build an even better model to fit the changing needs of the business. The process is never-ending, and it only becomes more complex as your business grows.

Enterprises investing in ML initiatives should consider implementing a machine learning DevOps strategy. Machine learning DevOps (MLOps) is an organizational change that relies on a combination of people, process, and technology to deliver machine learning solutions in a robust, scalable, reliable, and automated way.

This whitepaper overviews best practices for MLOps with Azure Machine Learning. It covers the technical capabilities of Azure ML and how it supports model reproducibility, validation, deployment, and monitoring. But just as importantly, we've thought through the key considerations and recommendations for improving processes

"We're delivering key new innovations in Azure Machine Learning that simplify the process of building, training, and deployment of machine learning models at scale"

Scott Guthrie

Executive Vice President of Cloud and AI

and making it easier for teams to collaborate. Ultimately, the goal of MLOps–your goal–is to close the gap between development and production and deliver value to customers faster. At Microsoft, our goal is to provide you with the best tools–tools that support your organization's processes and people wherever you are on your ML journey.

Understanding the typical ML lifecycle

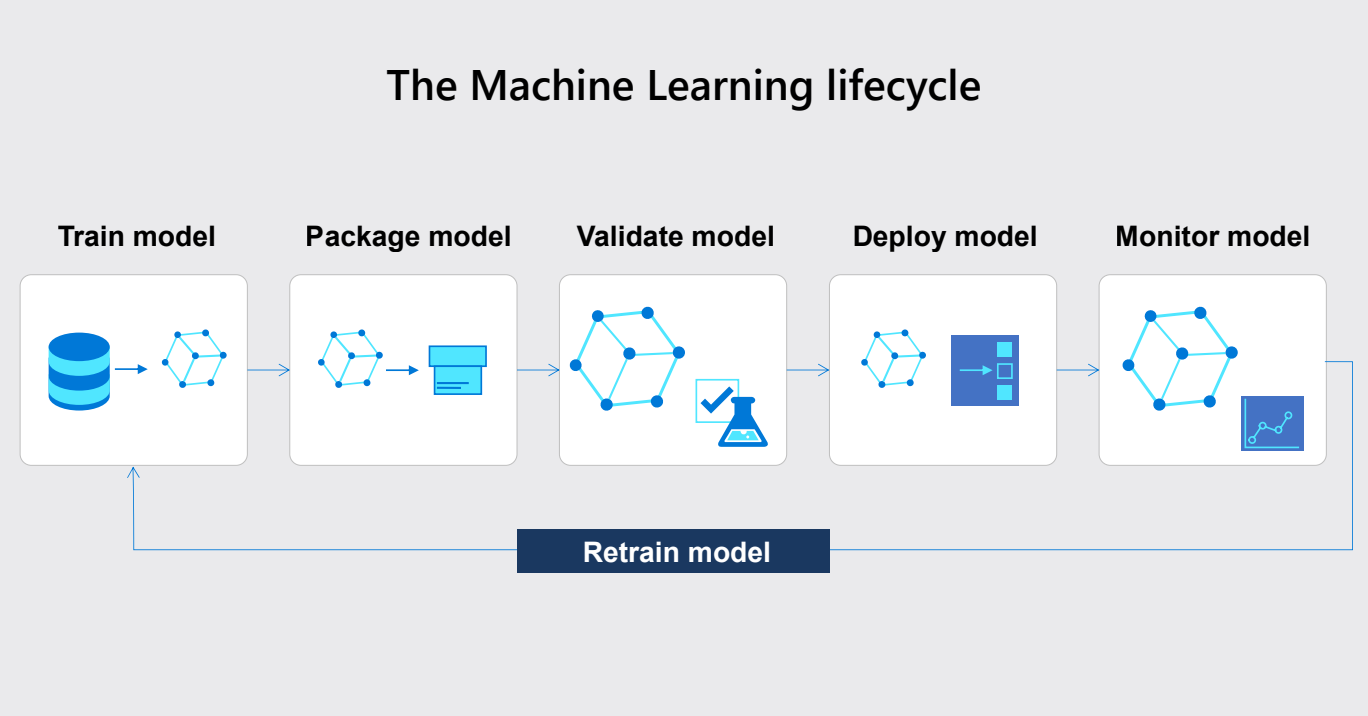
Creating machine learning models—taking models from raw data to a deployed model—involves multiple teams and roles and a wide range of tasks. Below is the flow of a standard ML lifecycle:

1. **Train and test:** First, data scientists need to prepare training data. This is often the biggest time commitment in the lifecycle. Preparation includes standardizing the data so it is in a usable format and identifying discrete "features" or variables. For example, to predict credit risk, features might include customer age, account size, and account age. Next, they apply algorithms to the data to "train" a machine learning model. Then they test it with new data to see how accurate its predictions are.

2. **Package:** ML engineers containerize the model with its environment, which means creating a docker container for the model to run in with all its dependencies. The model environment includes metadata like code libraries that the model needs to execute seamlessly.

3. **Validate:** At this point, the team evaluates how model performance compares to their business goals. For example, a company might want to optimize for accuracy over speed.

    - **Repeat steps 1-3:** It can take hundreds of training hours to find a satisfactory model. Data scientists may train many versions of the model by adjusting training data, tuning algorithm hyperparameters, or trying different algorithms. Ideally the model improves with each round of adjustment. It is the data scientists' role to determine which version of the model best fits the business use case.



4. **Deploy:** Finally, they deploy the model to a scale-out inference cluster in the cloud or on-premises. Once deployed, these models can be called via a REST endpoint to score/inference on new data. If needed, they can also be deployed to edge devices to make it work for low-latency scenarios, or disconnected/offline inferencing needs.

5. **Monitor and retrain:** Even if a model works well at first, it needs to be continually monitored and retrained to stay relevant and accurate. This is important because the model or the data may have drifted from its original definition or purpose. Additionally, the deployed infrastructure (like a Kubernetes cluster) may need scaling to improve latency or adhere to changes in the number of queries, for example.

Managing the entire lifecycle at scale is complicated and requires large amounts of time and resources. Without repeatable processes, data scientists must reinvent the wheel each time they create and deploy a new model. Siloed teams impede workflow alignment and collaboration, and critical aspects unique to ML—like the importance of model versioning, the duality code and data that is unique to ML, and concept drift—impact the deployment and usefulness of models.

Many enterprise AI and ML projects rely on traditional software development processes (DevOps) to address these challenges. Although many of the stages are the same, the uniqueness of ML model creation, deployment, and monitoring presents additional complexity and considerations:

- **Code and dataset management:** The source code of the model training process has limited value if it is not also accompanied by the dataset or datasets that were used to create the trained model.

- **Auditability:** It can be difficult to ensure that models meet regulatory standards and performance thresholds over time.
- **Traceability:** It can also be difficult to trace the result of an inference created in a production environment all the way back to the source code and training data sets used to build a model.
- **Explainability:** Black box models make it difficult to understand how the model works.
- **Quality assurance:** Extensive quality checks on both trained models—for interoperability, fairness, and accuracy—and deployed models—for data drift and performance issues—can be exceptionally challenging.

To address these challenges with machine learning, organizations need an approach that brings the agility of DevOps to the ML lifecycle. We call this approach MLOps.

Accelerating your end-to-end ML lifecycle with MLOps

At its core, MLOps is a set of processes and tools that allow data scientists, ML engineers, and app/solution developers to collaborate and increase the pace of model development. It is not a product you deploy but a process you engage with at a depth according to your organization's needs. Like DevOps for app development, MLOps introduces automation and traceability throughout the entire model lifecycle—from data prep and experimentation to deployment and model monitoring.

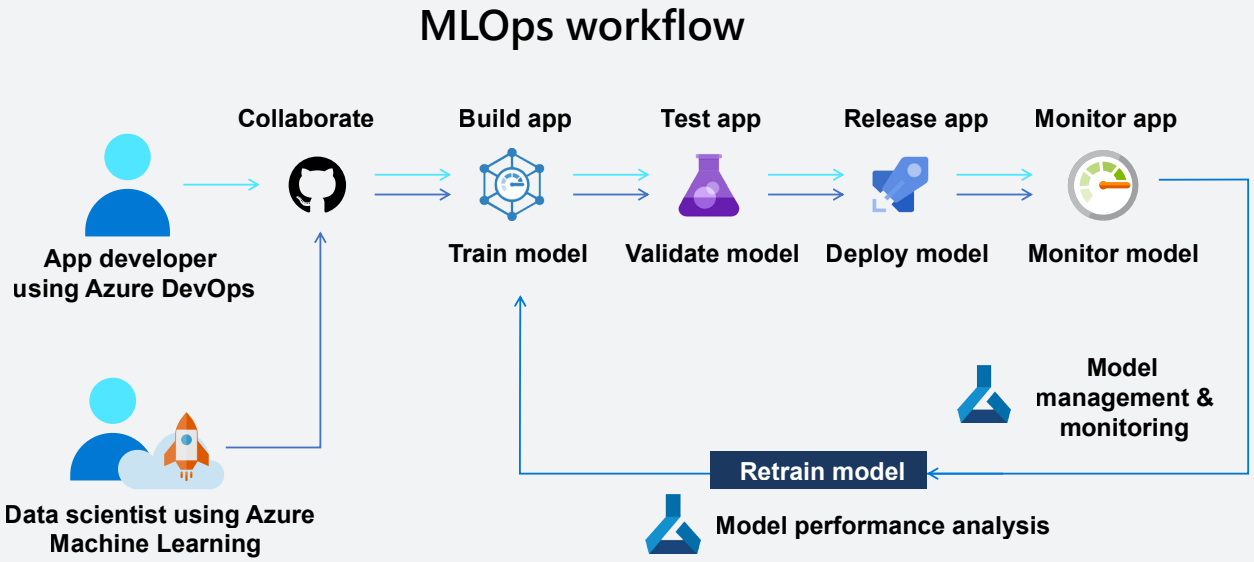
The benefits around MLOps are well-understood. Processes and tools simplify collaboration between data scientists, ML engineers, and app developers. They facilitate quality assurance and scalability—so your ML initiatives can grow with your business—and provide governance and structure to ensure auditability and fairness. Most importantly, they create efficiencies, accelerating the lifecycle with automation, repeatable workflows, and reusable assets, reducing the time from model creation to production deployment from months to days.

But while enterprises are rushing to embrace MLOps, the reality is that implementing a robust MLOps environment is itself a continuous improvement process. Increasing your company's maturity level when it comes to MLOps takes time, patience, and strategic thinking. Below are some best practices and key considerations when it comes people, process, and technologies, helping take your MLOps initiative progress to the next level.

Model reproducibility

Model reproducibility—the ability to produce models again and again—is at the heart of MLOps. The tools and processes that help design and automate repeatable and reusable steps along the machine learning lifecycle also help teams collaborate and iterate faster.

The process for achieving model reproducibility involves building a machine learning pipeline. Building a model—also known as experimentation—involves



many steps that need to be addressed and revisited, including feature selection, algorithm selection, hyperparameter tuning, fitting the model, and more. The process is essentially trial and error until you find a model that works and is suitable for the business objective. Azure ML enables you to identify independently executable workflows of machine learning tasks and save them to a pipeline that are visible to other members of your team. Importantly, when you want to reproduce that model, it is not just the model that gets picked up from a version history—it is the entire pipeline that is getting reproduced to accelerate future iterations.

Another key aspect of reproducibility is experiment tracking and asset integrity. As teams expand from a handful of people to tens and even hundreds of collaborators, being able to share an environment file and container, code repository, and registered models is critical. Without experiment tracking, there is no traceability for how the model was made, no ability to easily reproduce the model, and no governance data, such as who published the model and why changes were made. The ability to share and reproduce not just models, but all of a model's dependencies, helps teams collaborate at every stage in the cycle, troubleshoot in a targeted, efficient manner, and avoid duplicative work and re-doing costly and time-intensive steps. In addition to tracking artifacts, it's also important to track the performance KPIs used in machine learning experiments. When you keep a history of job performance, it allows for a quantitative analysis of experimentation success, which also enables greater team collaboration and agility.

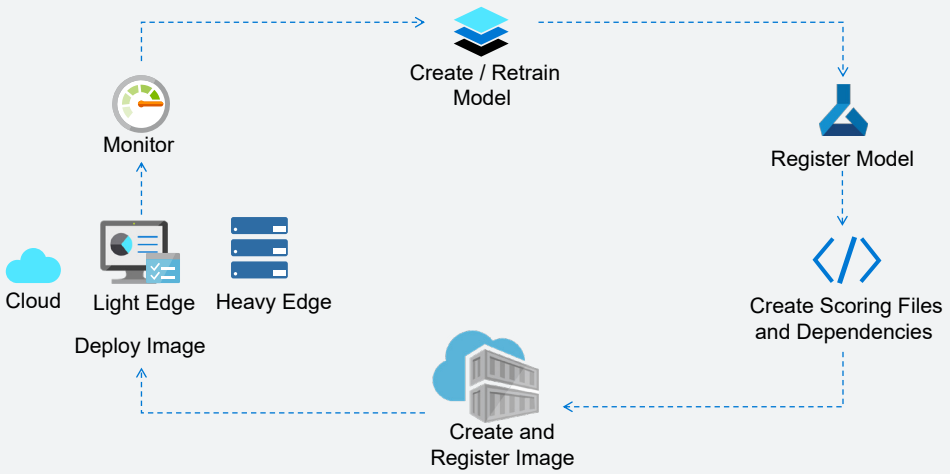
Azure ML supports model reproducibility with advanced tracking of datasets, code, experiments, and environments in a rich model registry. It can automatically produce an end-to-end audit trail of all your ML assets using metadata. Especially as enterprises scale and model creation velocity

increases, storing and managing models in a standardized manner is key to reducing both duplicative work and the computational cost of recreation.

There are several aspects to consider:

- **Model registry:** As teams experiment with different versions of a model, a model registry provides a central place to save each version. With a registry, teams can easily revert to a previous version if something is not working, even after the solution has gone into production. The model registry also serves as an audit trail for each model's history and makes it possible to automatically trigger workflows after certain actions or events. The Azure ML Model Registry captures all the metadata associated with your model (i.e., which experiment trained it, where it is being deployed, or if its deployments are healthy).
- **Source control:** You will also need technologies and processes for code management. Git is a popular version control system that allows you to save, version, share, and reuse code. Azure ML fully supports Git repositories for tracking work. You can clone repositories directly onto your shared workspace file system, use Git on your local workstation, or use Git from a CI/CD pipeline.
- **Dataset management:** We also recommend saving training datasets centrally. This helps you draw a lineage from the model to the underlying datasets. Teams can also reuse them, share them with colleagues, or monitor how they change over time to manage drift. Azure ML datasets help you track, profile, and version data so you can share and collaborate with other users, as well as seamlessly access data during model training without worrying about connection strings of data paths.

Model management



Step 1:
Register Model using the Model Registry

Step 2:
Register Image using the Image Registry (Azure Container Registry)

Step 3:
Deploy the image to cloud or to edge devices

Step 4:
Monitor models: You can monitor input, output, and other relevant data from your model

- **Shared environments:** Creating model environments that can be shared among individuals simplifies the handoff between steps in the model creation process and makes it possible for teams to collaborate on certain steps. Azure ML Run history stores a snapshot of the code, data, and computes used to train a model.
- **Automatic audit trails:** Enable automatic audit trails for all artifacts in your MLOps process to ensure asset integrity and meet compliance. Azure ML provides extensive support for managing machine learning-specific artifacts like experiments, runs, models, compute resources, models, images, deployments, and activities.

Of course, there are other tools—familiar to many data scientists—that cover some of the same functions. MLflow is a powerful open-source tool getting wide attention for simplifying management of the learning lifecycle, including experimentation, reproducibility, deployment, and model registry. The good news is, consistent with Microsoft's commitment to open source tools, you can easily connect Azure ML as the backend of your MLflow experiments. For practitioners comfortable with MLflow's UI, APIs, and other core features, there is no tradeoff involved—MLflow will reflect everything back to Azure ML, which can remain the version of truth on the backend.

Azure Machine Learning and MLflow integration

Model validation

Before a model is deployed, it's critical to evaluate whether it will do a good job predicting the target on new and future data. There are several aspects to model validation: does the model work from a technical standpoint? Does it deliver value against the business use case it was designed to support? And is the model fair—does it unintentionally reinforce biases, safety and privacy gaps, and exclusionary practices?

It's important to understand what metrics are being used to consider models successful. Accuracy alone is often not good enough to determine the overall performance of one model versus another. Validation also includes evaluating performance metrics against the business use case. Additionally, understanding whether the underlying infrastructure will support the use case is critical: for example, evaluating model response latency (queries per second) to be able to adjust the model's underlying infrastructure (such as Kubernetes cluster size).

The validation process brings different teams and experts together to come at the model from all angles to determine if it will work or fail. What can be a messy process is greatly improved with some key DevOps practices. Creating a pseudo-production environment and independent testing stages to cover all the different aspects of validation helps isolate testing and results so you can more easily pinpoint why a model might fail. This step-by-step evaluation of the model also helps create a clear set of roles and responsibilities for various roles and types of tests, driving efficiencies, and, ultimately, quality.

Here are some best practices to influence your validation tools and processes:

- **Unit testing:** The model is broken down into a small set of inputs and each input is tested separately and expected to produce stable results. Unit testing must often be followed by integration tests to make sure the components work together. While this is a crucial form of testing to ensure the model, it does not replace evaluating whether the model will perform under real-world conditions.
- **Validation and testing datasets:** Evaluating the performance of a model is one of the core stages in the data science process. It indicates how successfully the model will make predictions on data it hasn't seen before. We recommend reserving a validation dataset so the results can help optimize hyperparameters. A final testing dataset, which should not be labeled, can then be used to validate predictions and confirm the model was trained effectively.
- **Evaluating accuracy:** Azure ML supports both model evaluation and cross-validation to measure performance. The Evaluate Model measures the accuracy of a trained model's predictions based on a set of industry-standard metrics, including classification models, regression models, and clustering models. When you want to test the validity of your training set and model, cross-validation partitions the data into folds and then tests multiple models on

combinations of folds. By comparing the accuracy statistics for all the folds, you can interpret the quality of the data set and understand whether the model is susceptible to variations in the data.

- **Model package validation:** Because models themselves do not get deployed—they are containerized alongside their dependencies (libraries, frameworks, etc.) and the container is deployed—it is important to validate the model package to ensure that the container is sufficient to take to any production environment.

Prior to deployment, this is also the time to try to identify potentially harmful issues like biases, safety and privacy gaps, and exclusionary practices. The model may be accurate and even work well in a business context, but what if it won't serve certain other types of demographic data or cross-sectional populations?

Identifying consumer trends with Azure Machine Learning

"We've used the MLOps capabilities in Azure Machine Learning to simplify the whole machine learning process. That allows us to focus more on data science and let Azure Machine Learning take care of end-to-end operationalization."

Michael Cleavinger,

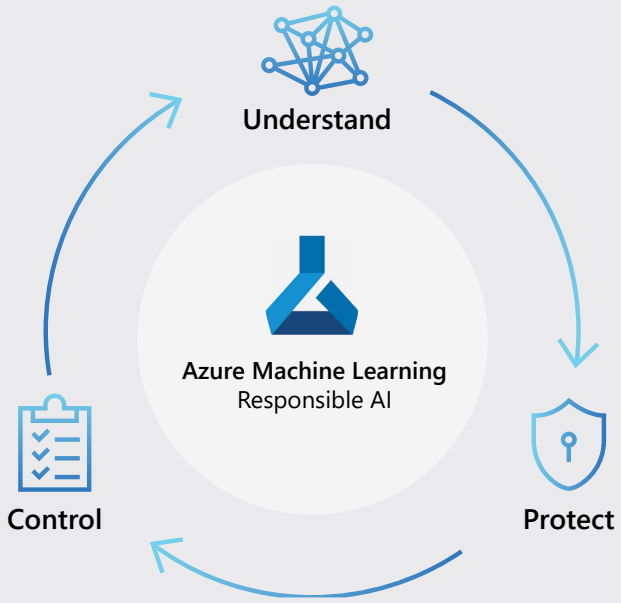
Senior Director of Shopper Insights Data Science and Advanced Analytics

Another goal is to ensure interpretability, which allows you to explain your models, meet regulatory compliance, and understand how models arrive at a result for given input. Tools alone do not achieve fairness and interpretability—it is vital to have strong governance, robust processes for monitoring for data drift, data sets that reflect diversity, and training so data scientists understand how bias can be introduced to the training processes. However, there are several resources that teams can use to assess models:

- **Assess fairness:** Fairlearn is an open-source Python package for assessing a system's biases and mitigating observed issues. It was designed for systems that make decisions about allocating resources, opportunities, or information. It has an assessment dashboard to understand how a model's predictions impact different groups. It also has mitigation algorithms to mitigate unfairness in binary classification and regression.
- **Increase transparency:** InterpretML is an open-source package created by Microsoft Research for training interpretable models and explaining black box systems. It implements a number of intelligible models including Explainable Boosting Machine (EBM), an improvement over generalized additive models that has both high accuracy and intelligibility. It also supports several methods for generating explanations of black box model behavior or predictions including 'SHapley Additive exPlanations' (SHAP) and 'Local Interpretable Model-agnostics Explanations' (LIME).
- **Achieve interpretability:** Azure Machine Learning has a variety of tools that support model transparency. The Model Interpretability feature enables designers and evaluators to explain why a model makes the predictions it does, which can be used to debug the model, validate that its behavior matches objectives, and check for bias.

For more on responsible machine learning best practice, read our companion whitepaper here.

Responsible AI with Azure Machine Learning



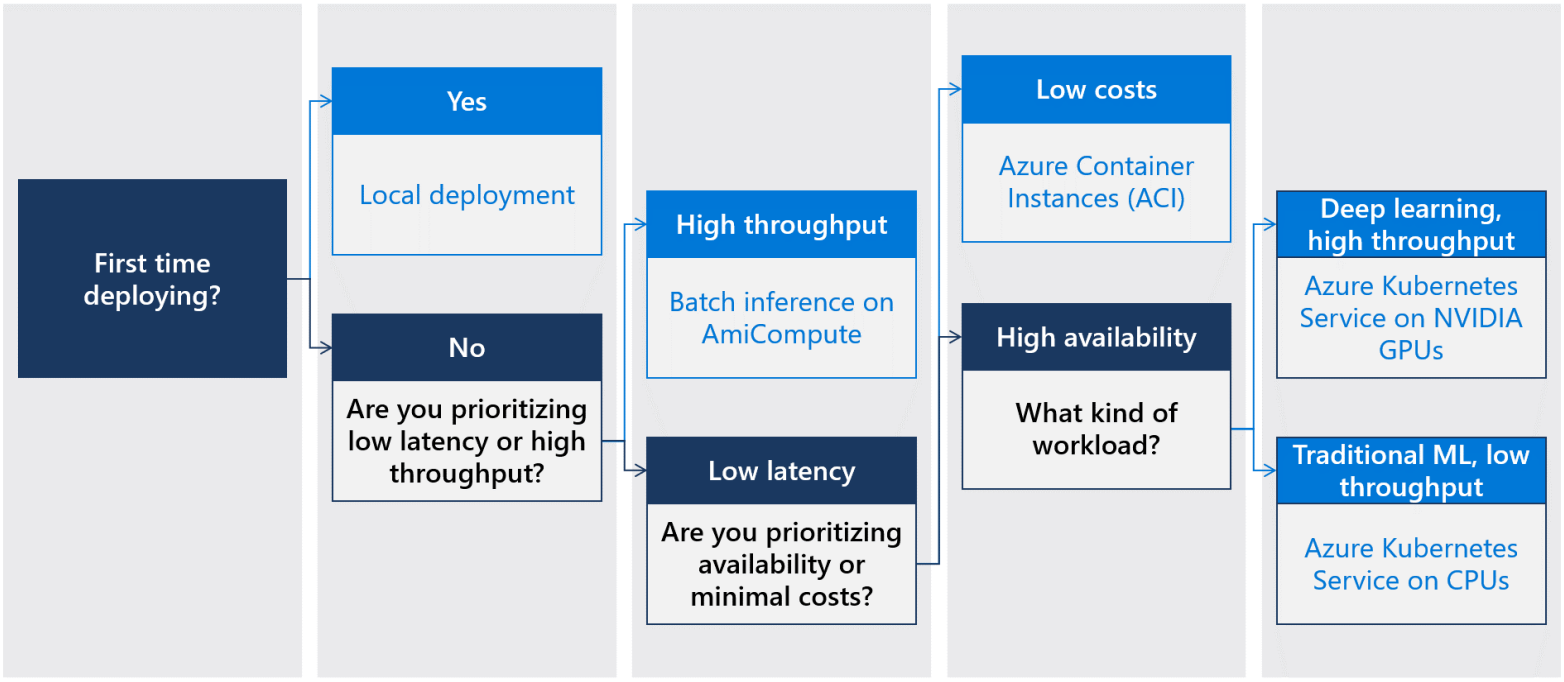
Model deployment

Data scientists should work with ML engineers and app developers to determine how to best deploy the model into production. No matter where you deploy the model (cloud, on-prem, edge devices), the workflow is similar. First, you'll register the model in the model registry. Then, you'll prepare to deploy the model by specifying assets, usage, and the compute target. Finally, you'll deploy it to your desired location, test it, and continue to monitor model-specific metrics throughout the lifecycle.

This stage, however, can be one of the largest pain points when it comes to cross-team collaboration, especially between data scientists, data engineers, and app developers. Data engineers must ensure that the infrastructure can support the model; for example, that the query throughput matches the way the model was designed. It is also hard to get a model baked into an app or service. There are often no gates or checks for shipping a functional/performance regression, and at the end of the day, staging and release are manual and time-consuming processes.

The goal of deployment best practices is to make models easy to deploy in an automated, predictable, and secure manner across a wide variety of inferencing targets: a standalone containerized inferencing service, part of a batch processing pipeline, or embedded in an app or service. A good CI/CD pipeline for models needs to expose simple user interfaces to make it easy for engineers to deploy and monitor models with minimal configuration. It also allows for a QA phase where models can be tested in a low-cost environment, followed by

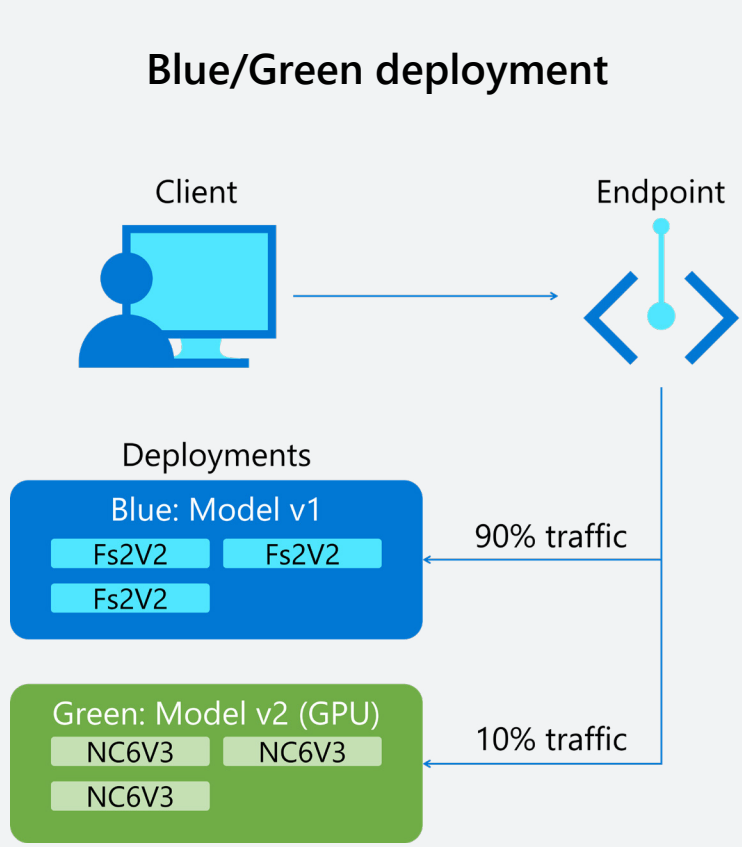
Choosing an inferencing target



a gated release. Ultimately, the goal is that someone doesn't always have to write a new script for a new deployment. Furthermore, it also needs to help users with various levels of machine-learning expertise understand and analyze their data and models.

Best practices for model deployment include:

- **Ease of use:** Simplifying model deployment is the top-line goal. Provide a friendly format for models (such as PMML or ONNX); simplify the process to interact with the model through code-generation, API specifications or other methods; support a variety of inferencing targets out of the box (cloud/app/edge), including specialized hardware such as FPGAs or dedicated frameworks (such as CoreML and WinML); and provide secrets and service endpoint management to remove friction from configuring the release process.
- **Govern the gating process:** Governing the release process of your models can be a daunting process: you'll need to ensure access controls, develop automated checks, and provide auditability of the deployment process. This is particularly true



when dealing with compliant release environments and customer data. To complement automated checks, the gating process can also benefit from human intervention–or "human-in-the-loop"–whereby manual reviews can certify models are ready for deployment by applying an expert lens. You use resource tags in Azure ML to indicate asset compliance, candidates for review, or as triggers for deployment.

- **Choose the right compute target:** The compute target you use to host your model will affect the cost and availability of your deployed endpoint. For example, a local web service is fine for limited testing and troubleshooting, but for high-scale production deployments that provide fast response times and autoscaling, consider Azure Kubernetes Service (AKS). Azure Container Instances supports low-scale CPU-based workloads where you don't have to manage a cluster, and Azure Machine Learning Compute Cluster supports batch scoring on serverless compute. Finally, managed endpoints in Azure ML help deploy models in a turnkey manner across powerful CPU and GPU machines in Azure in a scalable, fully managed way. These take care of serving, scaling, securing, and monitoring your ML models, freeing you from the overhead of setting up and managing the underlying infrastructure. Azure ML has capabilities for online/real-time scoring of models, as well as capabilities for simplifying the batch inference experience.
- **Production testing:** As part of the deployment process, we recommend testing the model in the context of your application or web service. Although offline validation prior to deployment may demonstrate adequate model performance on historical data, they do not always guarantee performance in a real-world setting. Azure Machine Learning automatically deploys and tests the packaged model on a variety of CPU inference and memory configurations to determine the optimal performance profile and checks that the inference service is responding correctly to these types of queries.
- **Blue/Green deployments for safe rollout:** A blue/green deployment technique provides two identical production environments. This is used when you have an existing model deployed in production and you want to deploy a new version. You can introduce the new model to production by rolling out the change to a small subset of users/requests before rolling it out completely. If the new model is successful, you can divert all traffic to a new ("green") environment and delete the old ("blue") environment. This is similar to A/B testing, a technique used to compare multiple variations of the same model in production to determine which one is more effective. You typically run A/B tests to get statistically valid measures of effectiveness, and this can help ensure you select the right model and tune the hyperparameters to best meet the business use case.

- **Versioning and storage:** Provide a consistent way to store and share models and track where models are embedded and running. You can version both experiments and trained models with Azure Machine Learning Studio: an immutable experiment snapshot is accessible using Run History, and a trained model is serialized into a format know as an iLearner file and saved in Azure Blob Storage.

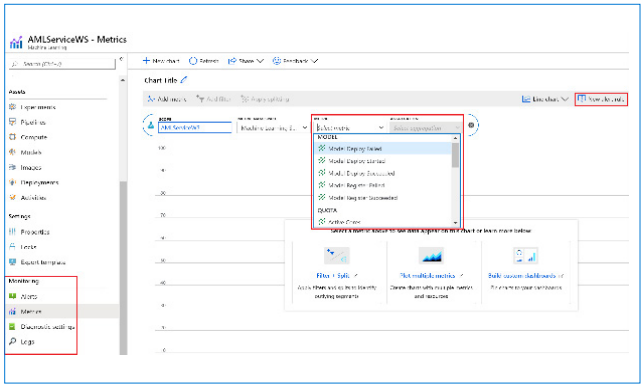
Model retraining

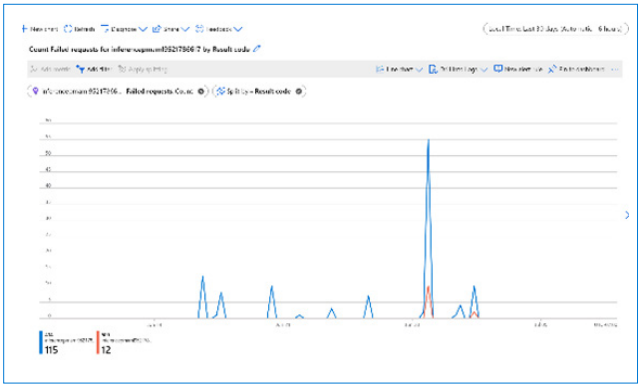
Although this is the end of the development process, it is just the beginning of the maintenance cycle. Models need to be monitored and periodically retrained to correct performance issues and take advantage of newer training data. To set yourself up for success you'll want to create a retraining loop: a systematic and iterative process to continually refine and ensure the accuracy of the model.

Identifying the right moment to retrain a model for production is not a trivial task. Retraining too often means disruptions for systems that rely on the model and potentially no significant performance improvements. Not retraining often enough means potential degradation to the performance of the model.

Knowing when to retrain models requires constant monitoring. So what exactly are you looking for? It is important to set up technical metrics to keep an eye on model performance; for example, query throughput and infrastructure. You are also analyzing model drift to see how the model changes– one of the top reason model accuracy degrades over time. There can be conceptual model drift where the business needs evolve and model becomes less suitable, and data drift where the input data changes (think demographic changes, seasonal changes) and leads to performance issues. Automating the monitoring process is key

Monitoring ML models with Azure Machine Learning





Monitoring in Azure Machine Learning has 23 metrics enabled that you can chose from across various categories like model, resource, run, and quota.

How to bring ML to production

People

- Blend together the work of individual engineers in a repository.
- Each time you commit, your work is automatically built and tested, and bugs are detected faster.
- Code, data, models and training pipelines are shared to accelerate innovation.

Process

- Provide templates to bootstrap your infrastructure and model development environment, expressed as code.
- Automate the entire process from code commit to production.

Platform

- Safely deliver features to your customers as soon as they're ready.
- Monitor your pipelines, infrastructure and products in production and know when they aren't behaving as expected.

as the process is ongoing, and provisions for alerting can help trigger retraining pipelines.

Best practices for monitoring and training include:

- **Monitor data drift from the input data**. With Azure Machine Learning model data collection, both production model input data and model predictions can be collected for monitoring purposes. Conceptually, there are three primary scenarios for monitoring in Azure Machine Learning: (1) monitoring a model's serving data for drift from training data, (2) monitoring time series data for drift from a previous time, and (3) performing analysis on past data. One way to monitor data drift is through Azure Application Insights, which provides an alert that can trigger actions like email, SMS text, push, or Azure Functions.
- **Analyze all collected data.** After saving data in Azure Blob Storage, the collected data is made available to query and analyze in a tool like Power BI. Make sure to collect data from models in production and include the results in the model scoring script. Collect all features used for model scoring, as this ensures that all necessary features are present and can be used as training data.
- **Decide whether retraining with the collected data is necessary**. Many things cause data drift, including sensor issues to seasonality, changes in user behavior, and data quality issues related to the data source. Model retraining isn't required in all cases, so it's recommended to investigate and understand the cause of the data drift before pursuing this. Use predefined criteria to choose whether to replace your old model.

- **Retrain the model.** Model training should already be automated and this step involves triggering the current training step. This could be for when data drift has been detected (and it isn't related to a data issue), or when a data engineer has published a new version of a dataset. Depending on the use case, these steps can be fully automated or supervised by a human. For example, while some use cases like product recommendations could run autonomously in the future, others in finance would factor standards like model fairness and transparency and require a human to approve newly trained models.
- **Bring in a human touch.** Human-in-the-loop is the process of bringing human intervention into the machine learning lifecycle, especially in the training and testing phases. Human experts can confirm, reject, or label outputs so that the model can learn from human inputs and improve its accuracy. Integrating the process into MLOps is mutually beneficial: human experts benefit from efficiency of the operations and models become better trained, ultimately giving humans more confidence in AI.

MLOps maturity model

	1 DevOps, No MLOps	2 Automated Training	3 Automated Model Deployment	4 Full MLOps Automated Retraining
Training	Untracked, file is provided for handoff	Tracked, run results and model artifacts are captured in a repeatable way	Tracked, run results and model artifacts are captured in a repeatable way	Tracked, run results and model artifacts are captured in a repeatable way, retraining set up based on metrics from app
Release	Manual, hand-off	Manual release, clean handoff process, managed by SWE team	Automated, CI/CD pipeline set up, everything is version controlled	Automated, CI/CD pipeline set up, everything is version controlled, A/B testing has been added
Integration	Manual, heavily DS driven	Manual, heavily DS driven, basic integration tests added	Semi-automated, unit and integration test added, still needs human signoff	Semi-automated, unit and integration test added, may need human signoff
People	Siloed	Mix of siloed and cooperative	Mostly cooperative with some siloing	Fully cooperative

Automating MLOps processes

At first, it's common for an organization to only automate a model's training and deployment but not the validation, monitoring, and retraining steps, which are performed manually. Eventually, automating steps for these tasks can progress until the desired state is achieved. DevOps and machine learning operations are concepts that develop over time and organizations should be aware of their evolution.

As companies start implementing MLOps, the first step in the process is generally some form of DevOps and is often the implementation of continuous integration/continuous deployment (CI/CD) pipelines. The continuous integration approach helps measure and control the quality of your team's work and accelerates time-to-feedback on experimentation. Continuous deployment for machine learning models automates the deployment and testing of real-time scoring services across your Azure environments (development, test, production).

But while DevOps brings some level of automation (automated tests for code, for example), much of the "ML awareness" comes through the implementation of custom code and disparate tools. GitHub Actions has a series that integrates parts of the data science workflow with the software development workflow. It is the default tool to create a CI process that trains a model, as well as a CD process that deploys the model as a web service. There are a growing number of actions available for MLOps with GitHub Actions, including Orchestrating Machine Learning Pipelines, Jupyter Notebooks, End-to-end Workflow Orchestration, Experiment Tracking, and more. GitHub Actions integrates seamlessly with Azure ML, so in a typical scenario where a data scientist checks a change into the Git repo for a project, Azure Pipelines will start a training run. The results of the run can then be inspected to see the performance characteristics of the trained model.

Focusing on automation is a key aspect of growing your company's MLOps capabilities. Fewer incidents or errors will lead to improvements in the quality of the development and production processes, and helps with auditability and explainability.

By accelerating and simplifying the model lifecycle, data scientists, data engineers, and IT professionals using ML pipelines can focus on high-value, innovative tasks.

Following implementation of DevOps and CI/CD pipelines, companies often focus on automating model training and reproducibility. From there the next step is automating model deployment, including integrated A/B testing for model performance. A mature MLOps environment is fully automated and easily monitored, including testing, monitoring, and retraining.

Even in stages where DevOps has been implemented, data scientists, data engineers, and software engineers are often still siloed. The first step to connecting data scientists with data engineers is to convert experimentation code into repeatable scripts and jobs. Next is to pull in software engineers to work with data engineers to automate model integration into application code. And finally, in a fully automated, mature environment, the teams are collaborating on multiple aspects of the pipeline.

Establishing MLOps at organizational scale

As the number of use cases grows in an organization, the management burden of supporting these use cases grows linearly, or even more. The challenge becomes how to use organizational scale to accelerate time-to-market, quicker assessment of use case feasibility, enable repeatability, and how to best utilize the available resources and skillsets across the full range of projects.

Building and deploying enterprise-ready AI solutions means adopting a strategic approach to ML—one that streamlines execution at scale and takes into account security and compliance, governance, and responsibility. For more on how to scale your operations and infrastructure, incorporate security and compliance best practices, and think about governance, read our whitepaper on enterprise readiness here.

Conclusion

Not every organization's machine learning DevOps (MLOps) requirements are the same. The MLOps architecture for a large, multinational enterprise is unlikely to fit a small startup. Organizations start small and build up as their maturity, model catalog, and experience grows.

Microsoft aims to meet organizations where they are on their ML/AI journey. Our leading technologies and robust MLOps capabilities can help you accelerate the machine learning lifecycle and empower data scientists and developers to build, train, and deploy models on a secure, trusted platform that supports a wide range of productive experience and is designed for responsible machine learning. To learn more, visit the Azure Machine Learning Operations page.

Additional resources

Infosheet

MLOps Best Practices

Learn best practices for extending DevOps practices to Machine Learning pipelines on Azure.

Demo

MLOps Demo

Watch this interactive demo which showcases reproducible ML pipelines, deployment, CI/CD, and governance of ML Projects.

Paper

Standardizing the ML Lifecycle

Find out how to solve for ML lifecycle challenges compared to the traditional software development lifecycle.

eBook

Mastering Azure Machine Learning

Explore this free eBook from Packt for hands-on guidance, real examples, and executable code on Azure ML.

eBook

Principles of Data Science

Get a comprehensive beginner's guide to statistical techniques and theory.

Paper

Four Real-Life Machine Learning use cases

Dive into four practical end-to-end machine learning use cases on Azure Databricks.

Microsoft

© 2021 Microsoft Corporation. All rights reserved. This whitepaper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT. The descriptions of other companies in this document, if any, are provided only as a convenience to you. Microsoft cannot guarantee their accuracy, and the companies and products may change over time. Also, the descriptions are intended as brief highlights to aid understanding, rather than as thorough coverage. This document is provided "as is." Information and views expressed in this document, including URL and other Internet websitereferences, may change without notice. You bear the risk of using it. This document does not provide you with any legal rights to any intellectual property in any Microsoft product.